MA 3046 - Matrix Analysis
Laboratory Number 8
Iterative Improvement and Numerical Accuracy

As an algorithm for solving the system of linear equations

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \quad , \quad \mathbf{A} \in \mathbb{C}^{m\times m} \tag{1}$$

Gaussian elimination has numerous attractive features. First, it's simple, easily understood and easily implemented, especially for the hand calculations required in introductory courses. In addition, from a realistic computational standpoint, it's very efficient, in terms of both space requirements and computational complexity. Furthermore, it is very attractive, both analytically and computationally inasmuch as it can be viewed as equivalent to factoring the matrix $\mathbf{A}$ into a product of two simpler matrices, i.e.

$$\mathbf{A} = \mathbf{L}\,\mathbf{U} \tag{2}$$

where $\mathbf{L}$ is lower triangular, and $\mathbf{U}$ is upper triangular. Moreover, even incorporating row interchanges (partial pivoting) as a strategy to minimize error growth produces only a minor and almost equally simple to obtain variant of the factorization, i.e.

$$\mathbf{P}\,\mathbf{A} = \mathbf{L}\,\mathbf{U} \tag{3}$$

where $\mathbf{P}$ is a permutation matrix. Finally, once we have this factorization in hand, then solving (1) simply requires solving the two equations

$$\begin{aligned} \mathbf{L}\ \mathbf{z} &= \mathbf{P}\,\mathbf{b} \\ \mathbf{U}\,\mathbf{x} &= \mathbf{z} \end{aligned} \tag{4}$$

Therefore, since each of the systems in (4) is already triangular, each of these two equations can actually be solved in about n^2 flops, with no need for elimination - a cost negligible when dealing with systems of any real size compared to the approximately $\frac{2}{3}n^3$ flops required for elimination on the original matrix $\mathbf{A}$. (Also, multiplication by $\mathbf{P}$ is in fact only a rearrangement of subscripts, not a computation!) Unfortunately, this savings is not available the first time we need to solve the system for a particular right-hand side, since we need to generate the $\mathbf{P}$, $\mathbf{L}$ and $\mathbf{U}$ factors in the first place, and that costs as much as elimination, since it is, in fact, just elimination. However, in applications we commonly need to solve the same basic problem, i.e. the same matrix $\mathbf{A}$, but with a number of different right-hand sides, i.e. we need to solve

$$\mathbf{A}\,\mathbf{x}^{(k)} \;=\; \mathbf{b}^{(k)} \quad , \quad k = 1, 2, \ldots$$

For such systems, if we utilize the **PLU** decomposition, we must pay the full $\frac{2}{3}n^3$ cost **only once**, i.e. only for $k = 1$. Then, if we simply save $\mathbf{P}$, $\mathbf{L}$ and $\mathbf{U}$, all subsequent solutions cost only $2n^2$, i.e. they are essentially free.

Hopefully though, by now in this course, we recognize that computational efficiency is not the sole criterion by which we must evaluate algorithms, and we are have learned to view all theoretical claims from an at least slightly jaundiced viewpoint of how well they actually play out when implemented for real matrices, on real computers using floating-point arithmetic. In addition, we have also already seen that the primary characteristic which separates "good" from "bad" algorithms as far as accuracy is concerned is backward stability. For the $\mathbf{PA} = \mathbf{LU}$ factorization, backward stability would mean that

$$\frac{\| \tilde{\mathbf{P}}\mathbf{A} - \tilde{\mathbf{L}}\tilde{\mathbf{U}} \|}{\| \mathbf{A} \|} = \mathbf{O}\left(\epsilon_{\text{machine}}\right)$$

where the tildes denote the computed solutions using finite-precision arithmetic. Unfortunately, the best result we can show is that

$$\frac{\| \tilde{\mathbf{P}}\mathbf{A} - \tilde{\mathbf{L}}\tilde{\mathbf{U}} \|}{\| \mathbf{A} \|} = \mathbf{O}\left(\rho\, \epsilon_{\text{machine}}\right) \tag{6}$$

where the constant ρ is called the growth factor. Even more unfortunately, the best we can also show for ρ is

$$\rho \leq 2^{m-1}$$

and m does not need to be very large before $\rho\epsilon_{\text{machine}} = \mathbf{O}(1)$, i.e. before, for all *practical* purposes, we lose backward stability on any realistic machine. Fortunately, matrices having actual growth factors this large exist (at least **so far**) only in pathological mathematical examples, and most actual matrices seem to have growth factors satisfying $\rho \leq \sqrt{m}$. (But we may only have been lucky so far!) Finally, even for matrices for which the growth factor is "normal," it is still possible that $\mathbf{A}$ is ill-conditioned. In this case, no algorithm can be expected to accurately solve the system. Therefore, as a practical matter, since Gaussian elimination and the $\mathbf{PA} = \mathbf{LU}$ factorization may in fact fail to accurately solve some problems, and since we frequently don't know ahead of time whether a system we're solving is "difficult" or not, we need to look for fairly inexpensive tests which will tell us at least whether a solution we've just computed is any good.

Again, here we are in luck. There are a couple of reasons for this. First of all, we already have developed a fundamental relationship, which we have previously discussed both in class and in a previous laboratory, that establishes the relationship between the *error*, i.e. the difference between the actual ($\mathbf{x}$) and computed ($\tilde{\mathbf{x}}$) solutions,

$$\mathbf{e} = \mathbf{x} - \tilde{\mathbf{x}} \ ,$$

and the *residual*,

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}\mathbf{e}$$

to be

$$\frac{\| \mathbf{e} \|}{\| \mathbf{x} \|} \leq \kappa(\mathbf{A}) \cdot \frac{\| \mathbf{r} \|}{\| \mathbf{b} \|} \tag{7}$$

where $\kappa(A)$ denotes the condition number ($\equiv \parallel \mathbf{A} \parallel \cdot \parallel \mathbf{A}^{-1} \parallel$) of $\mathbf{A}$ in any legitimate norm. This relationship allows us to take advantage of the fact that the residual is easily and cheaply computed, once we have found $\tilde{\mathbf{x}}$. Moreover, (7) clearly implies that a relatively "large" residuals virtually guarantee poor solutions. Unfortunately, as we have already seen, "small" residuals do not necessary imply correspondingly small errors. So what do we do when the residual is small?

The **PLU** decomposition can play a fundamental role in answering this last question, i.e. in determining the actual accuracy of computed solutions because, as we have seen, both the original solution and the error satisfy a system of equations with exactly the same left-hand side, i.e.:

$$\begin{aligned} \mathbf{A}\,\mathbf{x} &= \mathbf{b} \\ \mathbf{A}\,\mathbf{e} &= \mathbf{r} \end{aligned} \tag{8}$$

Therefore, one intriguing "fallout" of the fundamental inequality (7) is that, in general, we should expect to see similar *relative* errors in both our computed solution ($\tilde{\mathbf{x}}$) and in a computed *estimate* to the error ($\tilde{\mathbf{e}}$) based on solving the second equations in(8). Put somewhat differently, if $\tilde{\mathbf{x}}$ has at least the correct order of magnitude and one significant digit of accuracy, then solving the second equation should produce an error estimate of the correct order of magnitude and at least one significant digit of accuracy. Therefore, $\tilde{\mathbf{x}} + \tilde{\mathbf{e}}$ should have not only the correct order of magnitude, but at lease **two** significant digits of accuracy. Moreover, since both equations in (8) have the same left-hand sides, then, provided we solved the first using the $\mathbf{PA} = \mathbf{LU}$ decomposition, then solving the second for $\tilde{\mathbf{e}}$ is effectively free!

This can be extended into the so-called iterative improvement algorithm:

Starting with the original computed solution, $\tilde{\mathbf{x}}^{(0)}$

$$\begin{aligned} &\text{Compute :} && \mathbf{r}^{(n)} = \mathbf{b} - \mathbf{A}\,\tilde{\mathbf{x}}^{(n)} \quad , \\ &\text{Solve :} && \mathbf{L}\,\mathbf{z}^{(n)} = \mathbf{P}\,\mathbf{r}^{(n)} \\ & && \mathbf{U}\,\mathbf{e}^{(n)} = \mathbf{z}^{(n)} \\ &\text{Update the solution :} && \\ & && \tilde{\mathbf{x}}^{(n+1)} = \tilde{\mathbf{x}}^{(n)} + \tilde{\mathbf{e}}^{(n)} \quad , \end{aligned}$$

There is one major caveat in this approach. This method only works if the residuals are computed accurately. Since, a small residual can arise only if significant cancellation occurs when we subtract $\mathbf{A}\tilde{\mathbf{x}}$ from $\mathbf{b}$, then to ensure accuracy, the residuals must be computed:

- In double precision, and
- Using the original matrix $\mathbf{A}$, not $\mathbf{L}\,\mathbf{U}$

and repeating the process until either convergence is observed, or appears not possible.

We shall test this algorithm using several pre-written MATLAB function and script **m**-files that incorporate the (misnamed) **chop( )** command. The specific ones we shall use

```
function [ r ] = calc_resid_chop( a , b , x , NDIGITS )
%
  if  ( exist('NDIGITS')  = 1 )
       error(['   NDIGITS must be assigned a value', ...
              ' prior to calling calc_resid' ])
  end
%
 rowsA = size( a, 1 ) ;
%
 A = chop( a , NDIGITS ) ;
 B = chop( b , NDIGITS ) ;
 X = chop( x , NDIGITS ) ;
%

 for  i = 1 : rowsA
   S(i)  = chop( sum( chop(  A( i , :).*X' , NDIGITS ) ) , NDIGITS ) ;
 end
%
 r  = chop( B - S' , NDIGITS ) ;
return
```

Figure 8.1 - Partial Listing of Program **calc_resid_chop.m**

include **fpsolve_chop.m**, which we used earlier, and which uses the results of also earlier-used **lupp_chop.m**, and some other previously distributed programs to simulate the $\mathbf{P\,A} = \mathbf{L\,U}$ solution of given system in a simulated, low-precision machine; **calc_resid.m** (Figure 8.1), which simulates the calculation of a double-precision residual in a low-precision machine; and **itimp.m** (Figure 8.2), which simulates the iterative improvement process in a low-precision machine.

We would, however, be remiss if we did not mention before closing another variant of the fundamental inequality (7), which expresses the sensitivity of solutions to perturbations in the data. Specifically, if $\mathbf{b}$ is changed by a small amount, say $\boldsymbol{\delta}\mathbf{b}$, then we expect some resulting in the solution, say $\boldsymbol{\delta}\mathbf{x}$, where now

$$\mathbf{A}\,(\mathbf{x} + \boldsymbol{\delta}\mathbf{x}) \;=\; \mathbf{b} + \boldsymbol{\delta}\mathbf{b}$$

(Note by basic matrix rules then, $\mathbf{A}\,(\boldsymbol{\delta}\mathbf{x}) \;=\; \boldsymbol{\delta}\mathbf{b}$.) However, we can also show that:

$$\frac{\|\,\boldsymbol{\delta}\mathbf{x}\,\|}{\|\,\mathbf{x}\,\|} \leq \kappa(\mathbf{A}) \cdot \frac{\|\,\boldsymbol{\delta}\mathbf{b}\,\|}{\|\,\mathbf{b}\,\|} \tag{9}$$

```
%
   r = calc_resid_chop( A , b, x , 2*NDIGITS )  ;
%
   z = fwd_solve_chop( L , P*r ) ;
   e = bwd_solve_chop( U ,  z  ) ;
%
   niter = niter + 1 ;
%
   x = chop( x + e, NDIGITS) ;
   data = [ data ; niter  x' ] ;
```

Figure 8.2 - Partial Listing of Program **itimp.m**

and this may have serious repercussions, because while we may reasonably expect that $\| \mathbf{r} \| / \| \mathbf{b} \|$ would not be too far above the order of magnitude of machine precision, no such presumption can exist concerning $\| \boldsymbol{\delta} \mathbf{b} \| / \| \mathbf{b} \|$, at least not in the world of engineering! Moreover, we can also show that the change in solution due to perturbations of the matrix satisfies a similiar equation,

$$\frac{\| \boldsymbol{\delta} \mathbf{x} \|}{\| \mathbf{x} \|} \leq \kappa(\mathbf{A}) \cdot \frac{\| \boldsymbol{\delta} \mathbf{A} \|}{\| \mathbf{A} \|} \tag{10}$$

Taken together, these last two equations imply very strongly that simply because iterative improvement can accurately solving a fairly ill-conditioned problem may not be of much comfort, since, given component specifications, etc., that improved solution represent almost certainly only the accurate solution to the wrong problem! And if the problem is ill-conditioned, the small errors in $\mathbf{A}$ and $\mathbf{b}$ introduced by inexact components, etc., may mean the true solution to the real-world problem is a long way from the true solution to the mathematical problem we input to the computer.

We shall conclude this laboratory with a demonstration of this last property as well.

[This Page Intentionally Left Blank]

Name: ______________________________

MA 3046 - Matrix Analysis
Laboratory Number 8
Iterative Improvement and Numerical Accuracy

1. Login to your workstation and start MATLAB. Then, using any web browser, link to the course laboratories home page and download the programs:

calc_resid_chop.m and **itimp.m**

to your working space. Also make sure you still have the programs:

lupp_chop.m, **fpsolve_chop.m**, **fwd_solve_chop.m**, and **bwd_solve_chop.m**

plus the data file **lu_methods.mat**, used in earlier laboratories, are available in your work space.

2. Next, load the data file **lu_methods.mat**, then rename the matrix **A2** and the vector **b2** loaded there to **A** and **b**, respectively. Finally, clear **A1**, **A2**, **b1** and **b2** from memory, and check that you still have:

$$\mathbf{A} = \begin{bmatrix} 4.01 & 5.90 & 5.18 \\ -1.39 & -2.31 & -1.70 \\ 7.56 & 12.0 & 9.42 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 16.4 \\ -5.20 \\ 29.1 \end{bmatrix}$$

respectively. (Note the capital is essential here!)

3. Obtain the solution of $\mathbf{Ax} = \mathbf{b}$ for the matrix and vector in part 2 above problems using the MATLAB command $\mathbf{A}\backslash\mathbf{b}$.

$$\mathbf{xtrue} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}$$

4. Give the commands:

$$\begin{array}{l}\textbf{global NDIGITS}\\ \textbf{NDIGITS} \;=\; \textbf{5}\end{array}$$

What is "machine precision" with this value of **NDIGITS**?

Now solve the system, using the **fpsolve_chop m**-file.

$$\tilde{\mathbf{x5}} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Compare this answer to the one you computed in parts 3. How many correct digits does this solution appear to have? What, if anything, does that suggest?

5. Now, using full MATLAB precision, compute the residual

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\ \tilde{\mathbf{x5}} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

and the quantity

$$\frac{\|\,\mathbf{r}\,\|}{\|\,\mathbf{b}\,\|}$$

(You may do this either by hand, or using any of the MATLAB **norm(· ,)** commands.) How well does this value agree with the theory discussed in class?

6. Now give the command

NDIGITS=3 .

Then resolve the system again, using **fpsolve_chop**,

$$\tilde{\mathbf{x}}^{(0)} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Does what happens now seem consistent with the theory discussed in class? (Explain *briefly.*)

7. Again, using full MATLAB precision, compute the (new) residual

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\ \tilde{\mathbf{x}}^{(0)} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

and the quantity

$$\frac{\|\,\mathbf{r}^{(0)}\,\|}{\|\,\mathbf{b}\,\|}$$

Does this value still correspond to the theory discussed in class?

8. Make sure that the value of **NDIGITS** is still set to three, and also rerun **fpsolve_chop** again to make sure $\tilde{\mathbf{x3}}$ has not changed from part 6.

9. Study the MATLAB **m**-file **itimp.m**, which performs a *single iteration* of iterative improvement. (Note that the function of the semi-colons is to "turn off" the printed output of the values of some of the commands.)

10. Run a single iteration of **itimp.m** once and record the results:
Solution:

$$\mathbf{r}^{(0)} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}, \quad \tilde{\mathbf{e}}^{(0)} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}, \quad \tilde{\mathbf{x}}^{(1)} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}$$

What are the values of:

a. $\dfrac{\|\mathbf{r}^{(0)}\|}{\|\mathbf{b}\|}$ ______________________

b. $\dfrac{\|\tilde{\mathbf{e}}^{(0)}\|}{\|\tilde{\mathbf{x}}^{(0)}\|}$ ______________________

Are these values reasonable? Explain?

11. Repeatedly issue the command **itimp.m** until you feel the process has converged (based upon the behavior of either $\tilde{\mathbf{e}}^{(n)}$ or $\tilde{\mathbf{x}}^{(n)}$.

Solution:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix} \quad ; \text{ number of iterations } \underline{\qquad\qquad\qquad}$$

How well does the converged solution compare to the "exact" solution obtained in part 3? Should that have been expected?

12. Study program **calc_resid_chop.m** until you feel you understand what it's doing. Then edit **itimp.m** and change the statement:

calc_resid_chop(A , b , x , 2*NDIGITS)

to

calc_resid_chop(A , b , x , NDIGITS)

What effect should this have?

13. Be sure you have saved the changes you made just above. Then repeat parts 8-11, and repeat the iterative improvement at least **ten** times, recording the last three values of $\tilde{\mathbf{x}}$.

Solution:

$$\tilde{\mathbf{x}}^{(n)} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix} , \quad \tilde{\mathbf{x}}^{(n+1)} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix} , \quad \tilde{\mathbf{x}}^{(n+2)} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}$$

Do these values appear to be converging? If not, why not?

14. Edit **itimp.m** to now change the statement:

calc_resid_chop(A , b , x , NDIGITS)

to

calc_resid_chop(inv(P)*L*U , b , x , 2*NDIGITS)

What effect should this have?

15. After making sure you have saved these latest changes, repeat part 8-11. Again try iterative improvement at least **ten** times, and recording the last three values of $\tilde{\mathbf{x}}$.

Solution:

$$\tilde{\mathbf{x}}^{(n)} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}, \quad \tilde{\mathbf{x}}^{(n+1)} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}, \quad \tilde{\mathbf{x}}^{(n+2)} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Do these values appear to be converging? If not, why not? If they do appear to be converging, how well does the converged solution now compare to the "exact" solution obtained in part 3? Should that have been expected?

16. Replace the value of $\mathbf{b}(3)$ with 29.2, and solve the system again using the normal MATLAB backslash ($\mathbf{A}\backslash\mathbf{b}$) function.

$$\tilde{\mathbf{x}} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Considering that the normal MATLAB backslash command works in full (i.e. double) precision does this answer seem consistent with theory when compared to the answer to part 3? (Explain *briefly*!)